Section 8 Work Sheet (Solution)

Created by Wei Liao and Matt Xu

List of patterns: Singleton, Interning, Factory Method, Factory Object, Builder, Adapter, Decorator, Proxy

```
public class MysteryOtter {

    private static final MysteryOtter instance =
                                        new MysteryOtter();

    //private constructor to avoid client applications to use
constructor
    private MysteryOtter(){}

    public static MysteryOtter getInstance(){
        return instance;
    }
}
```

Q1: What design pattern is this?

Singleton (Specifically Eager Initialization)

```
public CuteCow CreateCuteCow(String variety){
      if (variety.equals("white"))
            return new WhiteCuteCow();
      if (variety.equals("shining"))
            return new ShiningCuteCow();
      if (variety.equals("rainbow"))
            return new RainbowCuteCow ();
      return new EmptyCow();
}
```

Q2: What design pattern is this?

Factory Method

```
public class MysteryFlamingo {

    private static MysteryFlamingo instance;

    private MysteryFlamingo(){}

    public static synchronized MysteryFlamingo getInstance(){
        if(instance == null){
            instance = new MysteryFlamingo();
        }
        return instance;
    }
}
```

Q3: What design pattern is this?

Singleton (Specifically Lazy Initialization)

```
public class MysteryCat {
    private final String cuteness, temper;

    private static Map<String, MysteryCat> instance =
                      new HashMap<String, MysteryCat>;

    private MysteryCat(String cuteness, String temper) {
        this.cuteness = cuteness;
        this.temper = temper;
    }

    public static Cat getInstance(String cuteness,
                                  String temper){
        String key = cuteness + temper;
        if(!instance.containsKey(key)){
            instance.put(key,
                         new MysteryCat(cuteness, temper));
        }
        return instance.get(key);
    }

    . . . . . .
}
```

Q4: What design pattern is this?

Interning

```
interface IceCreamFactory {
    IceCream getIceCream();
}

class VanillaFactory implements IceCreamFactory {
    public IceCream getIceCream() {
        return new VanillaIceCream();
    }
}

class ChocolateFactory implements IceCreamFactory {
    public IceCream getIceCream() {
        return new ChocolateIceCream();
    }
}

class StrawberryFactory implements IceCreamFactory {
    public IceCream getIceCream() {
        return new StrawberryIceCream();
    }
}
```

Q5: What design pattern is this?

Factory Object

Q6: Try writing a Builder class and a corresponding class **(TA check for answers)**

Q7: Which design pattern specifically requires the class be immutable?

Interning

Q8: Which design pattern enforces that only one object of the class can ever exist at runtime?

Singleton

(remember to explain to students why it is not interning, the difference is important)

Q9: Which design pattern uses an extra class to store properties needed by the constructors and eliminates the need for multiple constructors in the original class?

Builder

Q10: Matt made a program that can beautifully and perfectly visualize a list on his phone (in his dream). Now he wants to write a new and better one base on it. Match the following scenarios to the correct structural patterns:

Matt wants to visualize a list that is stored on the cloud but not on his phone **(solution: Proxy)**

Matt wants to make it also able to visualize HashMap and TreeSet on his phone: **(solution: Adaptor)**

Matt wants to make it also able to read content out loudly: **(solution: Decorator)**